WASP: Workload Agent-Based Simulation Platform for Migration Recommendations in Federated Kubernetes Environments

Andre Cunha[1], **Jose Lima**[1], **Lilia Sampaio**[1], **Giovanni Farias**[1], **Fabio Morais**[1]

[1]Distributed Systems Laboratory (LSD)
Federal University of Campina Grande (UFCG)
Campina Grande, PB – Brazil

{andre.cunha,jose.lima,liliars}@lsd.ufcg.edu.br

{giovanni,fabio}@computacao.ufcg.edu.br

Abstract. *Workload migration in federated Kubernetes environments is a complex task that requires well-defined migration strategies capable of operating under dynamic system conditions and constraints to balance performance, cost, and availability. However, enforcing them directly in production may lead to performance degradation, particularly under unvalidated autonomous agent-based operation. This paper presents WASP (Workload Agent-Based Simulation Platform), a decision-support tool that enables operators to simulate agent-based migration strategies before production deployment. WASP adopts a modular architecture with monitoring, recommendation, and execution control layers, supporting configurable policies with human-in-the-loop approval.*

1. Introduction

Managing workload placement and migration in federated Kubernetes environments has become a complex operational problem. Cloud-native applications must adapt to fluctuating workload demands, evolving resource availability, and heterogeneous infrastructure constraints. In hybrid environments, where private and public clusters coexist with distinct performance and cost profiles, migration decisions must account for dynamic system conditions while satisfying multiple operational objectives [Wang et al. 2020, Kamran and Nazir 2018].

In practice, workload migration decisions in Kubernetes clusters are largely performed manually, based on monitoring dashboards and operational expertise. However, this approach does not scale well as cluster size and workload diversity increase. Manual decision-making becomes harder to sustain and more prone to errors, particularly when balancing conflicting objectives such as resource utilization, performance isolation, and cost efficiency [Zhang and Boutaba 2014, Panda and Jana 2015]. These limitations motivate autonomous agent-based approaches capable of analyzing monitoring data and making migration decisions under multiple operational constraints.

Several works have explored automated or semi-automated workload migration strategies in cloud and containerized environments. More recently, agent-based reasoning and AI-driven techniques have been proposed to coordinate multiple objectives and operational constraints [Abdel Khaleq and Ra 2023, Sanjalawe et al. 2025, Dias et al. 2025,

Nedoshivina et al. 2024, Fawad 2023]. While such automation can improve responsiveness and scalability, fully autonomous migration strategies raise concerns regarding safety and operator trust, particularly in production environments. These concerns motivate human-in-the-loop (HIL) designs, in which agent-based systems generate migration recommendations while final decisions remain under explicit human control [Amershi et al. 2014, Amershi et al. 2019].

Another practical challenge lies in validating migration strategies before enforcing them in production clusters, where experimentation is costly and risky. Simulation and emulation environments provide a safer alternative, enabling controlled and reproducible evaluation of migration policies [Wen et al. 2023, Straesser et al. 2024]. Recent work indicates that lightweight Kubernetes simulation environments can approximate realistic workload behavior and cluster dynamics without deploying real containers, supporting iterative testing and refinement of migration strategies prior to production adoption [Lima et al. 2025].

In this paper, we present *WASP (Workload Agent-Based Simulation Platform)*, a decision-support tool for workload migrations in federated Kubernetes environments that preserves human oversight. WASP employs an AI agent-based reasoning engine to analyze structured, monitored data and generate workload migration recommendations, which are exposed to operator validation. The platform also supports configurable operation modes, enabling deployment under different operational policies.

WASP adopts a modular architecture composed of monitoring, recommendation, and execution control layers. The recommendation layer supports multiple configurations, including single-agent and multi-agent setups. Additionally, WASP integrates a simulation-based execution environment that enables controlled evaluation of agent-based migration strategies without requiring deployment on production clusters.

2. Related Work

Workload management in Kubernetes environments can be broadly categorized into policy-based mechanisms, AI-driven decision systems, and simulation-based frameworks. Native approaches rely on human-defined policies to control resource allocation, but do not explicitly address multi-cluster workload migration or multi-objective decision-making under dynamic conditions [Zhang and Boutaba 2014, Panda and Jana 2015]. To address these limitations, AI-driven strategies have been proposed for scheduling and resource management [Abdel Khaleq and Ra 2023, Sanjalawe et al. 2025, Dias et al. 2025, Nedoshivina et al. 2024, Fawad 2023], demonstrating the potential of data-driven and agent-based reasoning, though typically assuming fully automated execution without human-in-the-loop validation.

Simulation frameworks provide controlled environments for evaluating workload behavior without impacting production systems. Tools such as k8sSimulator (K8Sim) [LINC-BIT 2023] and lightweight simulation approaches [Wen et al. 2023] enable reproducible experimentation, but focus on predefined scheduling policies rather than dynamic decision generation. In contrast, WASP integrates monitoring, agent-based reasoning, validation, and execution into a unified pipeline supporting both automated and human-in-the-loop operation, enabling runtime generation and evaluation of workload migration recommendations within a simulation environment.

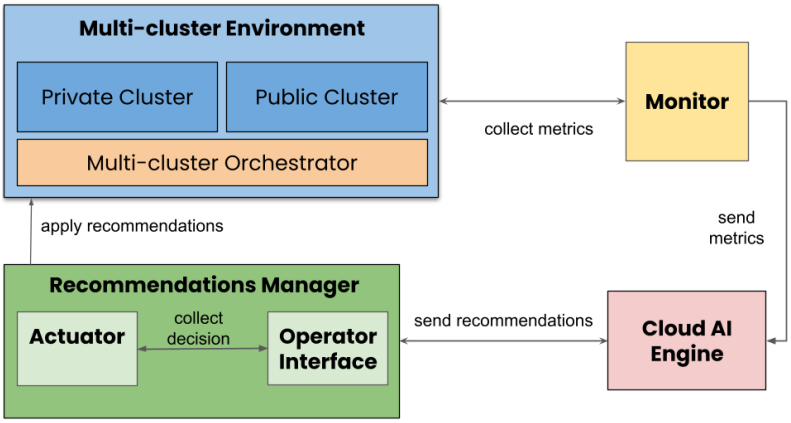

Figure 1. Overview of the WASP.

3. Architecture and Components

WASP adopts a modular architecture composed of three main components: Monitor, AI Engine, and Recommendations Manager. The Recommendations Manager is structured into two modules: an Actuator, responsible for enforcing migration decisions, and an optional Operator Interface. These components interact via explicit interfaces, forming a decision-and-control pipeline for workload migration in federated Kubernetes environments. This modular design allows components to evolve independently while maintaining a consistent operational flow. An overview of the system is illustrated in Figure 1.

The Monitoring component continuously collects cluster and workload metrics from target clusters at configurable intervals and provides structured information to the AI Engine. Based on these metrics, the AI Engine generates workload migration recommendations using configurable reasoning strategies, including heuristic-based approaches, machine learning models, or LLM-based agents.

Migration recommendations produced by the AI Engine are forwarded to the Recommendations Manager, which serves as the boundary between recommendation generation and execution control. The recommendations can be exposed to operators for inspection and approval or rejection, or directly forwarded to the Actuator for automated enforcement. This separation enables WASP to support both HIL and automated operation modes without requiring architectural changes.

3.1. Monitoring and Metrics Collection

The Monitor component collects cluster and workload metrics from Kubernetes clusters, forming the observability layer that feeds the AI Engine. Metrics are gathered at configurable intervals and provide contextual information about cluster state and resource utilization.

The monitoring layer integrates with the Kubernetes API to retrieve cluster-level metrics, node resource information, and workload status indicators (e.g., pod state and deployment health). From these inputs, it derives key indicators such as CPU and memory utilization, workload status (e.g., running, pending, failed), pending pod ratios, and cluster configuration attributes (e.g., autoscaling presence). Collected metrics are normalized

into a structured representation of the system state and stored for auditing in a MongoDB-backed layer. Data are serialized into JSON format for downstream processing.

The Monitor plays a strictly observational role within the WASP architecture: it collects and structures without performing analysis or triggering actions. This separation ensures that reasoning remains fully encapsulated within the AI Engine, while the monitoring layer provides a reliable and reproducible view of the system state.

3.2. AI Engine for Workload Migration Recommendations

The AI Engine is implemented as a Python-based service that transforms structured metrics into workload migration recommendations. It operates as an API service, enabling integration with external components.

Metrics from the monitoring layer are consumed through a typed interface that normalizes workload descriptors into a consistent representation of cluster state. This representation captures essential attributes, including workload identifiers, resource requests, execution status, cluster association, and temporal context, ensuring that the decision logic operates on a well-defined system view.

Based on this normalized view, the engine produces machine-actionable recommendations corresponding to concrete decisions, such as maintaining the current placement of a workload or migrating it to a different cluster. Outputs are structured to explicitly separate the decision from its justification, enabling reliable interpretation and traceability by downstream components. This design allows the recommendation logic to evolve independently from the execution and validation layers.

Finally, the AI Engine is designed to be model-agnostic. Interaction with underlying reasoning backends is abstracted through a uniform interface, allowing heuristic-based, learning-based, or LLM-driven strategies to be employed without modifying the surrounding architecture. This abstraction preserves interface stability while enabling flexibility in how migration decisions are generated.

3.2.1. Supported Reasoning Configurations

WASP supports multiple reasoning configurations that define how collected metrics are processed to generate workload migration recommendations. These configurations are exposed as alternative operational modes of the AI Engine and can be selected according to deployment context, operational preferences, and desired levels of control. Primarily, all modes preserve the same input/output interfaces, ensuring architectural consistency.

In the single-agent configuration, a unified reasoning component processes metrics in a linear cycle and produces migration decisions with associated explanations based on an explicit policy encoded in its prompt. This mode requires minimal setup, making it suitable for rapid experimentation and simple deployments.

Alternatively, WASP supports a multi-agent configuration in which reasoning is decomposed across specialized agents that independently analyze different aspects of the system, e.g., performance or cost. A consolidator agent synthesizes these partial assessments into a single migration recommendation. This structure enables potentially conflicting objectives to be considered independently while still producing a coherent decision.

3.3. Human-in-the-Loop Validation and Execution Control

WASP supports both fully automated execution and human-in-the-loop (HIL) execution modes for workload migration. In HIL mode, the AI Engine's migration recommendations are exposed through a web-based Operator Interface, allowing operators to inspect proposed actions and their justifications before execution. In automated mode, recommendations are directly enforced without human intervention.

All recommendations are handled by the Recommendations Manager, which separates decision-making from execution. Upon receipt, each recommendation is persisted in a database with explicit execution states (pending, approved, rejected, or expired). In HIL mode, operators may approve or reject pending recommendations through the Operator Interface. Recommendations are generated in batches, and unreviewed pending items are automatically marked as expired when a new batch arrives. This process prevents stale decisions from being executed under outdated system conditions. In all cases, decisions and associated explanations are recorded, ensuring traceability and a complete audit trail.

Execution is performed by an Actuator within the Recommendations Manager. In HIL mode, only recommendations marked as approved are enforced, while in automated mode recommendations are directly applied. This state-based gating mechanism decouples recommendation generation from execution and ensures that unapproved decisions do not affect the Kubernetes environments. Approved recommendations are executed through an orchestrator abstraction that translates structured migration plans into Kubernetes-compatible operations, such as updating placement policies or triggering workload propagation, while remaining independent of specific orchestration backends.

Additional safeguards, including staleness checks and feasibility validation, are applied before enforcement. By clearly separating recommendation generation, validation, and execution, WASP enables controlled, auditable, and reproducible workload migrations across both automated and supervised operational modes.

3.4. Implementation Stack and Tooling

WASP is implemented as a collection of modular services[1]. All components except the AI Engine are developed in Go [Golang 2026], leveraging its suitability for concurrent and networked system while ensuring compatibility with libraries and tooling widely adopted in the Kubernetes ecosystem. The AI Engine and its auxiliary modules are implemented in Python [Python 2026], enabling integration with AI libraries and flexible experimentation with different reasoning strategies. Persistent storage is provided by MongoDB [MongoDB 2026], which stores collected metrics, migration recommendations, and operator decisions. System configuration is managed declaratively through YAML files, supporting reproducible deployments without requiring code modifications.

To preserve model and provider agnosticism, WASP integrates OpenRouter [OpenRouter 2026] as a default abstraction layer for accessing LLM backends. This integration decouples the AI Engine from specific model providers, allowing heterogeneous models to be selected through configuration without modifying the surrounding architecture. As a result, different reasoning backends can be evaluated under consistent execution conditions.

[1]The tool is released as open source under the Apache License 2.0 and the source code is available at `https://github.com/cloud-ai-ufcg/simulator`

For experimentation and validation, WASP relies on a lightweight Kubernetes emulation environment based on KWOK (Kubernetes WithOut Kubelet) [KWOK 2026], integrated with Karmada [Karmada 2026] for federated orchestration. KWOK emulates cluster state and workload scheduling behavior without deploying real containers, significantly reducing resource requirements for running migration experiments while preserving Kubernetes control-plane semantics. Karmada manages federated workload propagation through label-based placement policies and policy synchronization mechanisms. This setup enables controlled and reproducible evaluation of migration strategies without requiring production infrastructure.

The Operator Interface is implemented as a web-based application using React [React 2026] and Vite [Vite 2026], communicating with backend services through RESTful APIs to support the inspection and validation of recommendations. All services are containerized, supporting consistent deployment across environments and independent extensibility.

4. Use Case

This section presents a representative workload migration scenario executed with WASP. We first describe the experimental setup, including cluster configuration, workload characteristics, and reasoning parameters, and then illustrate how the AI Engine generates migration recommendations that are validated through the HIL execution process.

4.1. Experimental Environment

The minimum hardware requirements to run WASP include 8 CPU cores, $16\,GiB$ of memory, and $100\,GiB$ of disk storage. The experimental environment was configured using Ubuntu 22.04.5 [Ltd. 2026], GNU Make 4.3 [Free Software Foundation 2020], Docker 28.3.2 [Docker Inc. 2025a], Docker Compose 2.36.2 [Docker Inc. 2025b], and Go 1.24 [Golang 2026].

Although WASP is designed to operate in federated Kubernetes environments, the experiments presented in this work rely on a simulated multi-cluster setup rather than a deployment on real federated infrastructure. This simulated environment reproduces key control-plane behaviors and workload dynamics, enabling controlled and reproducible experimentation without impacting production systems. The same architecture can be extended to real federated deployments in future work.

4.2. Scenario Configuration

The main configuration hub of WASP is defined in the file `simulator/data/config.yaml`, which specifies cluster configuration parameters, including the number of nodes and the CPU and memory capacity available per node. It also allows enabling or disabling the cluster autoscaler, determining whether a cluster may dynamically adjust its node count during execution. In the evaluated scenario, two federated clusters, *member1* and *member2*, are instantiated. Each cluster starts with two nodes, each providing 8 vCPUs and $16\,GiB$ of memory (16 vCPUs and $32\,GiB$ per cluster). The autoscaler is enabled only for *member2*, allowing dynamic cluster expansion during execution.

Monitoring telemetry is collected every 30 seconds, while the AI Engine reasoning cycle runs every 60 seconds. The configuration also specifies the reasoning backend

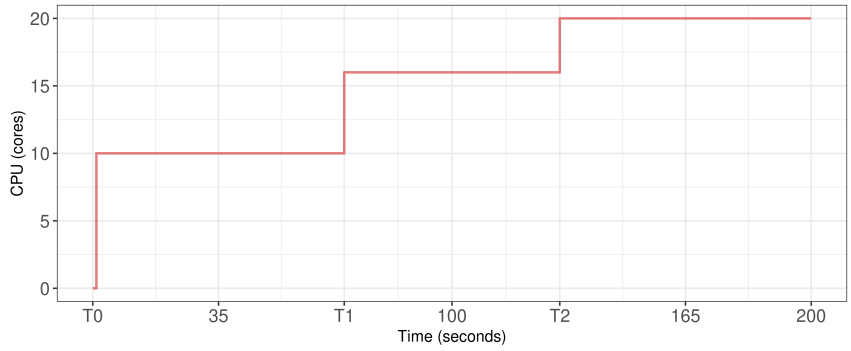

Figure 2. CPU requested by the submitted workloads over time.

and execution parameters, including the LLM provider, reasoning mode, and generation controls. In this scenario, the engine operates in single-agent mode using the OpenRouter provider, with temperature set to 0.1 to reduce the likelihood of hallucinations and increase determinism and *max_output_tokens* set to 18000. The system runs in HIL mode, requiring operator approval before migration actions are enforced.

Workloads are defined in `simulator/data/input.json` and injected into the environment by a broker service at predefined timestamps. Each workload event specifies the Kubernetes resource type (*kind*), number of replicas (*replicas*), CPU (*cpu*) and memory (*memory*) requests, initial cluster placement (*label*), and submission time (*timestamp*). In this scenario, all workloads are deployments with two replicas, each requesting 1 vCPU and $2\,GiB$ of memory, and are initially placed in *member1*.

Migration decisions are guided by a user-defined prompt located in the `ai-engine/prompts/` module. In the evaluated configuration, the prompt enforces two operational constraints: cluster load must remain below 80% of total capacity, and proportion of pending workloads must not exceed 40% of the total workload count. These prompts can be modified to define alternative operational policies and decision criteria.

4.3. Execution and Observed Behavior

Workloads are submitted in three waves at timestamps 1, 70, and 130 seconds, followed by a final event workload submission at timestamp 200. This staged submission progressively increases resource demand in the *member1* cluster. Figure 2 illustrates the evolution of CPU requested over time.

Figure 3 shows the evolution of the CPU allocated across the two clusters, member1 and member2. Workloads are initially allocated exclusively in the *member1* cluster, which is reflected in the steady increase of CPU allocation in *member1* while its capacity remains constant. After the first wave, allocation stabilizes at a higher level. The second wave at 70, as indicated by $T1$ in Figure 3, it increases again. And, the third wave at 130 seconds, marked by $T2$ in the same figure, cumulative demand approaches the cluster's capacity limit, reaching the highest allocation level observed.

With the monitor and reasoning intervals configured to 30 and 60 seconds, respectively, the AI Engine begins recommending migrations to *member2* once resource pressure approaches defined thresholds. Between 120 and 150 seconds, migration recommendations are approved, and the next monitoring cycle reflects the redistribution of

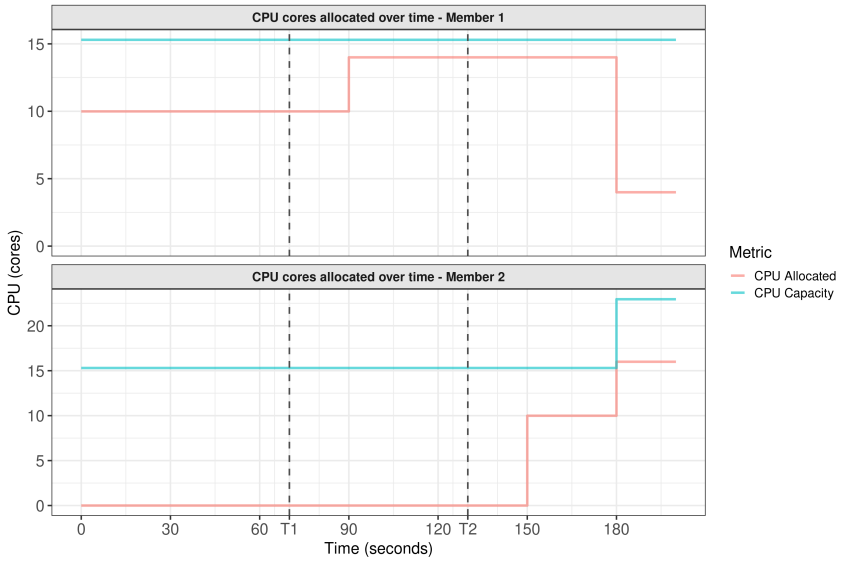

Figure 3. CPU requested over time from *member1* to *member2*.

workloads: CPU allocation in *member1* remains stable, while excess deployments are moved to *member2*. Additional migrations between 150 and 180 seconds further reduce pressure on *member1*.

In HIL mode, these recommendations appear in the Operator Interface as pending actions (see Appendix A, Figures 4 and 5). The operator may inspect the justification for each migration and either approve or reject the proposed action. Approved actions are forwarded to the Actuator, which enforces migration through placement policy updates and propagation mechanisms.

Together, these steps illustrate the complete WASP workflow, where workload arrivals, telemetry monitoring, reasoning, operator validation, and enforcement interact to manage resource pressure across clusters.

5. Conclusion

This paper presented WASP (Workload Agent-Based Simulation Platform), a decision-support tool for workload migration in federated Kubernetes environments. WASP combines monitoring, agent-based reasoning, human-in-the-loop validation, and controlled execution within a modular architecture, focusing on recommendation generation rather than fully autonomous enforcement.

By integrating agent-based reasoning with explicit operator oversight, WASP enables migration decisions that consider multiple operational objectives while preserving accountability and operator trust. The human-in-the-loop model provides a practical balance between automation and control, addressing key concerns associated with fully autonomous workload migration in production environments.

The use case presented in this paper illustrates how WASP supports the analysis, validation, and controlled enforcement of migration recommendations. Future work includes extending the supported reasoning strategies, incorporating additional metrics and policy constraints, and improving integration with federated management frameworks.

References

Abdel Khaleq, A. and Ra, I. (2023). Intelligent microservices autoscaling module using reinforcement learning. *Cluster Computing*, 26(5):2789–2800.

Amershi, S., Cakmak, M., Knox, W., and Kulesza, T. (2014). Power to the people: The role of humans in interactive machine learning. *Ai Magazine*, 35:105–120.

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., and Horvitz, E. (2019). Guidelines for human-ai interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA. Association for Computing Machinery.

Dias, T. M. R., Ferreira, L., Fevereiro, D., Rosa, L., Cordeiro, L., and Fernandes, J. F. P. (2025). Cloud-native scheduling and resource orchestration: a deep dive into ai-driven approaches. *IFIP Advances in Information and Communication Technology*, pages 101–114.

Docker Inc. (2025a). *Docker 28.3.2*. [Accessed: Apr. 16, 2026].

Docker Inc. (2025b). *Docker Compose 2.36.2*. [Accessed: Apr. 16, 2026].

Fawad, E. a. A. (2023). Efficient workload allocation and scheduling strategies for ai-intensive tasks in cloud infrastructures. *Power System Technology*, 47:82–102.

Free Software Foundation (2020). *GNU Make 4.3*. [Accessed: Apr. 16, 2026].

Golang (2026). Golang: Build simple, secure, scalable systems with go. `https://go.dev/`. [Accessed: Feb. 10, 2026].

Kamran and Nazir, B. (2018). Qos-aware vm placement and migration for hybrid cloud infrastructure. *J. Supercomput.*, 74(9):4623–4646.

Karmada (2026). Karmada: Kubernetes armada. `https://karmada.io/`. [Accessed: Feb. 10, 2026].

KWOK (2026). Kwok: Kubernetes without kubelet. `https://kwok.sigs.k8s.io/`. [Accessed: Feb. 10, 2026].

Lima, J., Cunha, A., Pedroza, N. F., Sampaio, L., Farias, G., and Morais, F. (2025). Ai-driven workload migration framework for multi-cluster environments. In *2025 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1–8. IEEE.

LINC-BIT (2023). K8ssim: A kubernetes cluster simulator for scheduling optimization. `https://github.com/LINC-BIT/k8sSimulator`. Accessed: Feb. 10, 2026.

Ltd., C. (2026). *Ubuntu Linux*. [Accessed: Apr. 16, 2026].

MongoDB (2026). Mongodb: The world's leading modern database. `https://www.mongodb.com/`. [Accessed: Feb. 10, 2026].

Nedoshivina, L., Levacher, K., Fraser, K., Halimi, A., and Braghin, S. (2024). Ai-driven workload management in meta os. In *Proceedings of the 1st International Workshop on MetaOS for the Cloud-Edge-IoT Continuum*, MECC '24, page 14–20, New York, NY, USA. Association for Computing Machinery.

OpenRouter (2026). Openrouter: The unified interface for llms. `https://openrouter.ai/`. [Accessed: Feb. 10, 2026].

Panda, S. K. and Jana, P. K. (2015). A multi-objective task scheduling algorithm for heterogeneous multi-cloud environment. In *2015 International Conference on Electronic Design, Computer Networks Automated Verification (EDCAV)*, pages 82–87.

Python (2026). Python: Getting started. `https://www.python.org/`. [Accessed: Feb. 10, 2026].

React (2026). React: The library for web and native user interfaces. `https://react.dev/`. [Accessed: Feb. 10, 2026].

Sanjalawe, Y., Al-E'mari, S., Salam, F., and Makhadmeh, S. N. (2025). Ai-driven job scheduling in cloud computing: a comprehensive review. *Artificial Intelligence Review*, 58.

Straesser, M., Haas, P., Frank, S., Hakamian, A., Hoorn, A. v., and Kounev, S. (2024). Kubernetes-in-the-loop: enriching microservice simulation through authentic container orchestration. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 82–98.

Vite (2026). Vite: Next generation frontend tooling. `https://vite.dev/`. [Accessed: Feb. 10, 2026].

Wang, b., Wang, C., Song, Y., Cao, J., Cui, X., and Zhang, L. (2020). A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds. *Cluster Computing*, 23.

Wen, S., Han, R., Qiu, K., Ma, X., Li, Z., Deng, H., and Liu, C. H. (2023). K8ssim: a simulation tool for kubernetes schedulers and its applications in scheduling algorithm optimization. *Micromachines*, 14:651.

Zhang, Q. and Boutaba, R. (2014). Dynamic workload management in heterogeneous cloud computing environments. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–7.

A. Operator Interface Usage

This appendix provides a visual overview of the Operator Interface, highlighting the main interactions available to users during human-in-the-loop (HIL) operation. The presented screenshots illustrate how migration recommendations are inspected, validated, and managed within the system, complementing the workflow described in the main text.

Figure 4 presents the Operator Interface filtered by pending items, showing an AI-generated migration recommendation for the workload *default/batch-worker*, proposing its relocation from *member1* to *member2*. In addition, the interface displays the associated justification, indicating resource overload on the source cluster. From this view, the operator can directly approve or reject the recommendation. In contrast, Figure 5 illustrates the same recommendation after operator validation. In this case, the interface reflects the approved status, while preserving the original justification and displaying an approval indicator.

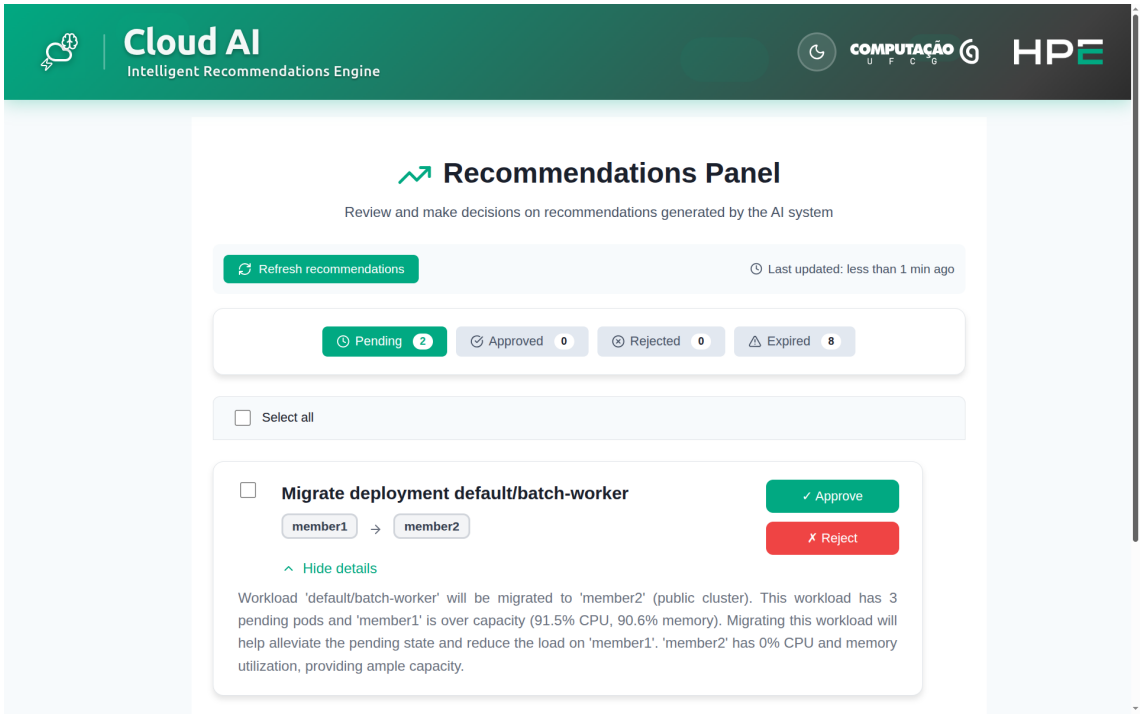

Figure 4. Pending recommendation awaiting operator validation.

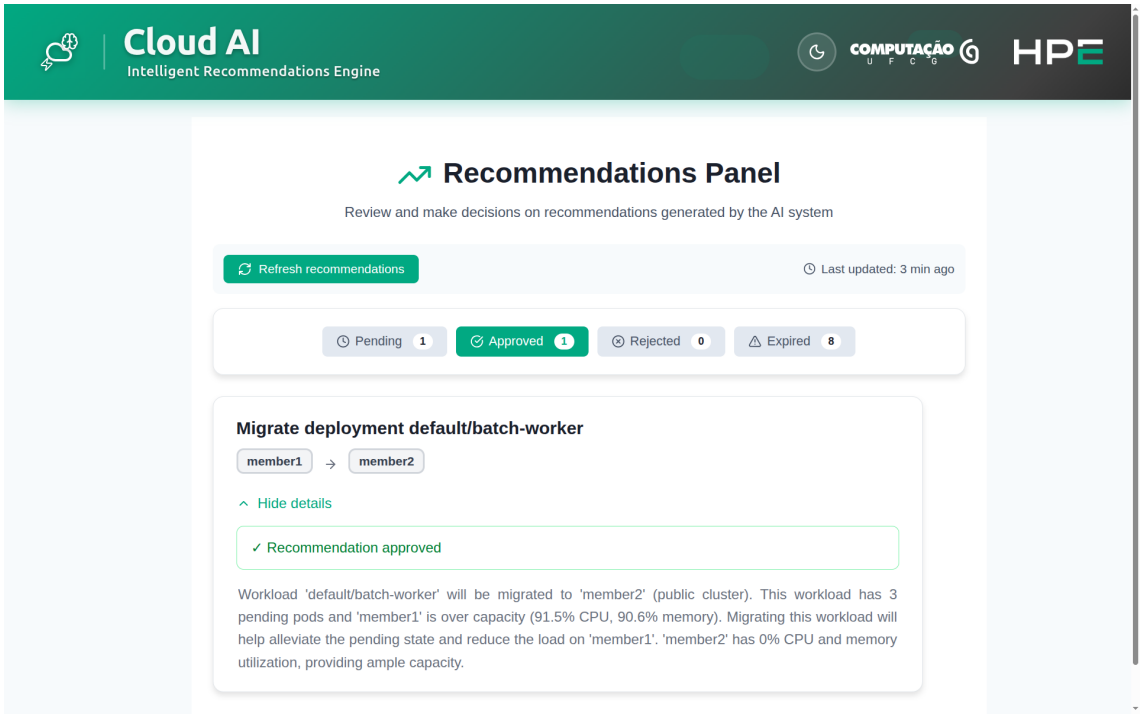

Figure 5. Approved recommendation after operator validation.